BELLCOMM, INC.

COVER SHEET FOR TECHNICAL MEMORANDUM

TITLE- A Note On Automatic Generation of Documentation by Macro Assemblers

TM- 64-1031-1

FILING CASE NO(S)- 210

DATE- September 2, 1964

AUTHOR(S)- W. M. Keese, Jr.

FILING SUBJECT(S)- (ASSIGNED BY AUTHOR(S))
Documentation
Automatic Documentation
Macro Assemblers

DRA

ABSTRACT

It is shown that macro assemblers can easily generate documentation, of a meaningful sort, which is, in at least one respect, superior to that which can be built into the 'self documenting', English-appearing languages such as ALGØL, NELIAC, etc.

Due to currently common limitations, however, this can be done only for very small programs. In particular, trivial difficulties with scan routines and unrealistic limitations on conditional assembly tend to contribute to the main problem of table overflow. It is found that an ALGØL-type table clean out feature would clear the way for practical use.

SEE REVERSE SIDE FOR DISTRIBUTION LIST

DISTRIBUTION

COMPLETE MEMORANDUM TO CORRESDONDENCE FILES:

OFFICIAL FILE COPY
PLUS ONE WHITE COPY FOR EACH ADDITIONAL CASE REFERENCED

REFERENCE COPIES

Bellcomm

Messrs. D. R. Anselmo
Mrs. C. H. Bronfin
B. F. Brown
B. M. Bruffey
J. O. Cappellari, Jr.
M. W. Cardullo
T. J. Celi
H. L. Davis
Miss P. B. Dogan
J. S. Dudek
Miss P. A. Egan
R. S. Farbanish
G. Findley
Miss L. G. Geiger
W. B. Gragg, Jr.
W. G. Heffron
D. C. Hobbs
J. E. Holcomb
J. A. Hornbeck
R. L. Ingle
M. S. Jones
Mrs. B. R. Kahan
Mrs. C. A. Keese
W. M. Keese
G. Kerbs
C. M. Klingman
J. Kranton
D. A. Levine
B. H. Liebowitz
Miss A. Locicero
H. S. London
C. A. Lovell
J. P. Maloy
W. J. Martin
Miss N. E. McConn
C. Mee
V. S. Mummert
I. D. Nehama

COVER SHEET ONLY TO CORRESDONDENCE FILES:

THREE COPIES-PLUS ONE COPY FOR EACH ADDITIONAL FILING SUBJECT

Complete Memorandum to continued:

Messrs. J. M. Nervik
M. J. Norris
Mrs. R. S. Orme-Johnson
E. B. Parker
J. E. Parker
W. H. Petty
H. Pinckernell
Miss G. E. Prather
H. E. Richards
A. L. Rothstein
A. A. Satterlee
J. A. Saxton
S. H. Schachne
A. H. Scheinman
J. J. Schoch
Mrs. N. Siehl
Miss K. Smith
A. E. Speckhard
Mrs. D. H. Speckhard
R. D. Rigner
T. H. Thompson
J. R. Vetter
J. M. West
M. S. Willoughby
T. C. Wood
D. C. Wurtzel

DISTRIBUTION

COMPLETE MEMORANDUM TO CORRESDONDENCE FILES:

OFFICIAL FILE COPY -
PLUS ONE WHITE COPY FOR EACH ADDITIONAL CASE REFERENCED

COVER SHEET ONLY TO CORRESDONDENCE FILES:

THREE COPIES-PLUS ONE COPY FOR EACH ADDITIONAL FILING SUBJECT

REFERENCE COPIES

BTL - Whippany

Messrs. P. V. Dimock
N. Foy
W. C. Johnson - 3 copies

CEIR

Messrs. L. P. Gieseler
R. D. Weiksner
R. T. Yuill

CUC

Messrs. W. Carlson
J. A. Fish
E. A. Kilroy

IBM

Miss J. Preble
Mr. J. M. Rimback, Jr.

NASA

Messrs. V. Huff
D. Turner

Philco

Messrs. J. L. Current
M. E. Ferguson
Mrs. M. C. Smith
A. I. Thaler

SHARE (25)

Yoh

Mr. J. Klingeman

SUBJECT: A Note on Automatic Generation of Documentation by Macro Assemblers - Case 210

DATE: September 2, 1964

FROM: W. M. Keese, Jr.

MEMORANDUM FOR FILE

Introduction

This note follows an interesting foray into automatic generation of meaningful documentation. The foray was undertaken in belief in the single document approach for a program, i.e., belief that a program listing, itself, should contain all of the documentation which the program requires* - including even the raw content of its various levels of flow charts.

This means that the source deck of a large program should contain many levels of documentation. There should be considerable explanation of the segmentation. Much material about data structures, good paragraphization within segments, etc. There is much of interest in automatic aids for this higher area, but this exercise is not concerned with higher levels. Rather, it addresses itself to the documentation of individual source instructions, particularly those of a higher level than simple arithmetic.

It is often the case that a good percentage of the source instructions in any large program are involved with decision making, calling of subroutines, transfers of control, and the simple moving of data. Somewhere, to be sure, there are routines which actually do things to data - which use arithmetic, Boolean, shift, etc. instructions. But, the larger the program is, the smaller part of it that these instructions seem to comprise.

In fundamental level documentation, no programmer needs to be told what (e.g.) a compare instruction does, but he does want to know what it is that is being compared. Simple machine knowledge allows recognition of a decision, but only an intimate

---

*This is not to say, however, that there should not be any other documentation.

knowledge of a program's data base may be able to provide knowledge of what decision is being made. Where subroutines are being called (or macros invoked), the situation is even worse, for no general knowledge is of any use in trying to follow the program. It can only be done if the call is accompanied by commentary that explains what is done.

This kind of commentary is vital to the feasibility of changing large programs and perhaps the most important factor in controlling the cost of such. Yet, experience tells us all that even the best of intentions are not always sufficient to assure that it is always correct and/or up-to-date.

When such documentation can be automatically generated, one is relieved of this large area of concern in programming control.

The enclosed example shows that it is easy, with a quite small macro package, to automatically generate good, meaningful commentary for the kinds of things outlined above - documentation about decisions and control, and explanation of the effect(s) of subroutines.

The possibility of this hinges upon the fact that all such actions can readily be performed by macros. These can, in fact, all be done with a standard macro package (see appendix). The use of such a package for decision and control is quite standard, and needs no explanation here. It is shown in section 1. that the calling of subroutines can and should be effected in a similar way, with no extra macro definitions on the part of the programmer. (An added advantage of treating all subroutines, in the source language of the calling program, as though they were macros is that proper modularity can be retained in the source program even when it must be violated in the actual machine program.)

An example of what can be done terminates this section. The actual macro package which creates it is included in the appendix.

Since this macro package is rather confusing when approached as a whole, an heuristic approach is used. Most of the main concepts arise in the development of the subroutine documentation macros, and this is done in some detail in sections 1. - 5. Sections 1 and 2 give the milieu from which the rest arose. Section 3 shows the generation of simple comments, and variable substitution is added in section 4. In section 5,

expansion is made to occur word for word on a recursive basis. The other macros used in the sample are briefly noted in section 6. Section 7 notes limitations and draws conclusions.

Particular note should be taken of the difference between the grade of documentation produced by the macros appearing in 3, 4, and 5. In 3, one gets nothing but the expanded subroutine name. In 4, however, one can obtain a meaningful sentence, with operands plugged into their proper (English sentence structure) places. This is at once a major improvement in documentation over functional notation, such as is found in compilers. In order to expand these operand names, so that the reader may fully understand the documentation without continual reference to the noun list, one has to go to the greater effort shown in section 5.

The various examples happen to be done in IBMAP for the 7040-44. In this, "IRP" followed by an argument, begins an indefinite iteration on the subfields of that argument. Its scope is ended by the next "IRP". "IFF" and "IFT" are conditioned assembly controls. With the first, the following card is assembled only if false that the (S) symbol value of the two expressions in the variable field are equal. With "IFT" the following card is assembled only if true that they are equal. Matched parentheses, in a macro instruction, are stripped, but make the entire character sequence which they enclose into a single argument. The apostrophe is simply the concatenation character.

```
*       ONE HEADS EACH ROUTINE OR SEGMENT WITH A 'USES' LIST,E.G.
*
        USES    APPND(CHAR)(APPEND,CHAR,TO,ID,STRING)
        USES    GNC()(GET,NC)
        USES    GNBC()(GET,NON,BLANK,CHARACTER)
        USES    SAVID()(SAVE,IDENTIFIER,STATUS)
*
*       WITH VARIABLE DECLARATIONS OF THE FORM
*
        VAR     NC(NEXT,CHARACTER)
        VAR     PC(PEEK,CHARACTER)
*
*               THE TEXT OF THE PROGRAM MAY THEN BE WRITTEN
*       IN ABBREVIATED FORM, AS BELOW, IN THE SOURCE DECK.
*
*       LABEL   ANAME(APOSTROPHE-ENCLOSED,NAME)
*       GNC
*       IF      NC,NEQ(=H00000')
*       THEN
*       GOTO    ANAME
*       ELSE
*       APPND   NC
*       GNC
*       GOTO    NONSPC
*       ...
*
*       LABEL   BLNKF(BLANK,FOUND)
*       APPND   PC
*       LAC     HEAD,I                  SET POINTER TO HEAD OF TREE.
*       GNBC
*       TRA     0,I                     TRACE SPECIAL WORD TREE.
```

```
                    •                   THIS WILL BE EXPANDED IN THE LISTING AS IS SHOWN HERE.
                    •
                    •
                          LABEL   ANAME(APOSTROPHE-ENCLOSED,NAME)
   00006          ANAME   EQU •
                                                  APOSTROPHE-ENCLOSED NAME ..
                          GNC
                                                    GET NEXT CHARACTER ,
                          IF      NC,NEQ(=H00000')
                                                  • IF NEXT CHARACTER NOT EQUAL TO
                                                    =H00000'
001
                          THEN
                                                  • THEN
                          GOTO    ANAME
                                                    GO TO APOSTROPHE-ENCLOSED NAME ,
                          ELSE
                                                  • ELSE
                          APPND   NC
                                                    APPEND NEXT CHARACTER TO ID
                                                    STRING ,
                          GNC
                                                    GET NEXT CHARACTER ,
                          GOTO    NONSPC
                                                    GO TO NONSPC ,
                          ...
                                                  • .....
002
                    •
                          LABEL   BLNKF(BLANK,FOUND)
      00016       BLNKF   EQU •
                                                BLANK FOUND ..
                          APPND   PC
                                                    APPEND PEEK CHARACTER TO ID
                                                    STRING ,
0 1 00003 10001           LAC     HEAD,I        SET POINTER TO HEAD OF TREE,
                          GNBC
                                                    GET NON BLANK CHARACTER ,
0 1 00000 10000           TRA     0,I           TRACE SPECIAL WORD TREE.
```

1. Background

The automatic documentation features described below were not created in vacua. Rather, they were recognized as a free bonus coming with certain other features being built into a set of standards for assembly language program packages.

Two of these are relevant. First: each routine or segment of a large program should carry with it a list of first level subroutines used. Second: while the fact that a subroutine is called is a part of the logical structure of the calling program, the way in which it is called is part of the logical structure of the subroutine - not of the calling program; it is therefore better modularity that the source language of the calling program should specify merely the existence, not the method, of the calls.

This second consideration gives rise to the notion that a particular call should be effected in an assembly language merely by writing the name of the subroutine in the operator code field (with any variable list(s) in the variable field). Particular knowledge of how a given subroutine is called can then be localized to one macro definition. Indeed, since, even in real time programs, most subroutines can be called in some standard manner, a standard call defining macro could be used to define most of the subroutines' calls.

The inclusion of a list of first order subroutines used, however, relieves us of even this necessity. This list, itself, can be used to define all the standard calls. Only those special routines which efficiency or special difficulty demand to have uncommon calls need have their names defined as call creating macros.

2. The Elementry Uses Macro

We were first led, then, to the creation of a USES macro. Each routine or segment of a large routine contains in its head a set of USES cards. Each of these cards contains the word 'USES' in its operator field and the name of a subroutine in its variable field, along with some explanation of the routine's effect.

The original purpose of these cards was simply the display of program organization. An elementry extension creates standard call definitions.

The USES macro first asks whether the subroutine name has been previously treated. If it has, it does nothing, but if it has not, then it defines it to create a standard system call. At this stage of development, the macro reads (in 7040 IBMAP)

```
USES    MACRØ   NAME
        IFF     NAME'.=DEFIND
        USES1   NAME
        ENDM    USES

USES1   MACRØ   NAME
NAME'.  SET     DEFIND
NAME    MACRØ   ARGLST,ERRT
        CALL    NAME(ARGLST)ERRT
        ENDM    NAME
        ENDM    USES1
```

(It is intended that this package run under no created symbols.)

The creation a symbol from "NAME" by concatenation of a period is possible for us, since the standards in use at Bellcomm limit the length of a subroutine name to one character less than maximum symbol length. Even without this, one could still ask whether definition had occured by making the meaning of the created NAME macro depend on whether or not some symbol was set to USES or to NØRMAL. Under the USES mode, NAME (if defined) would set the remainder of USES to be null (save for terminal self restoration). The remainder of USES would elsewise create the standard definition. This (using an operator-synonymous operator) could be done with no expandingly greater use of macro skeleton space, but it would be somewhat slower.

3. Generation of a Primitive Comment

Since rudimentary documentation practices demand that each of the USES cards carry some explanation, it is but a simple step forward to automatically reproduce this comment whenever the subroutine is called. This will be shown here in a primitive manner, while the next section will go on to show an elementry plugging in of arguments.

One merely encloses the explanation in parentheses (to make it a single argument) and moves it over from the comment field to adjoin the name. USES is now made a 2 argument macro. If it finds that NAME has not been defined, then it passes both the name and the (reparenthesized) comment along to USES1 which now reads

```
USES1   MACRØ   NAME,CØMENT
NAME'   SET     DEFIND
NAME    MACRØ   ARGLST,ERRT
        CALL    NAME (ARGLST)ERRT
        PMC     ØN
        REM                         'CØMENT'
        PMC     ØFF
        ENDM    NAME
        ENDM    USES1
```

In its definition of NAME, USES1 has added 3 card images. These turn on the printing of macro expansion cards, print the comment as a remark, and turn off the printing of macro generated cards: (as PMC is a control card, it, itself, will not be printed.)

In actual practice, NAME could be made shorter, saving skeleton space, by passing "CØMENT" along to another macro, which would do the three card expansion.

4. Comments With Substitutable Arguments

Whereas many subroutines have arguments, and their effect is on these arguments, a meaningful comment generator should plug the names of these arguments into the comments. This is a relatively easy addition.

In the following expansion, we start afresh and use slightly more levels, so as to conserve skeleton space. An argument list is added to the USES macro to make it possible to tell what words of the explanation should be substituted.

```
USES     MACRØ   NAME(ALIST)CØMENT
         IFF     DEFIND=NAME'.
         USES1   NAME(ALIST)(CØMENT)
         ENDM    USES

USES1    MACRØ   NAME(ALIST)CØMENT
NAME'.   SET     DEFIND
NAME     MACRØ   ARGLST,ERRT
         USES2   NAME(ARGLST)(ERRT)
         ENDM    NAME
NAME'.   MACRØ   ALIST,DUMMY1,DUMMY2
         PRSEQ   (CØMENT)
         ENDM    NAME'.
         ENDM    USES1

USES2    MACRØ   NAME(ARGLST)ERRT
         CALL    NAME(ARGLST)ERRT
         NAME'.  ARGLST,ERRT
         ENDM    USES2

PRSEQ    MACRØ   TEXT
         PMC     ØN
         REM                          'TEXT'
         PMC     ØFF
         ENDM    USES3
```

If one were not sure of the ability to concatenate the period to NAME, the macro "NAME'." could be named by a created label, generated by turning them on, going an extra level in, and turning them off. Its name would then be passed on to USES2.

We have reached the level which is best explained by an example, save for a preliminary note. While USES2 and PRSEQ

were created mainly* to conserve skeleton space (for each of the, presumably many, NAME and NAME'. macros), the creation of the "NAME'." macro serves a very different function. Since "ALIST" is built into its formal parameter list, while the comment is built into its skeleton, any delimited phrases in "CØMENT" which are the same as delimited phrases in "ALIST" will be substituted at expansion time. When a call is made (NAME is invoked) the actual parameter list is sent along to "NAME'." and the appropriate substitution is made for PRSEQ. The two dummy arguments of "NAME'." are merely to avoid a falacious diagnostic regarding argument count which 7040 1BSYS likes to make.

Suppose now that ADDTØ has not previously been defined and that the following source card appears.

```
USES   ADDTØ(A,B)(ADD 'A' TØ 'B')
```

This will expand into

```
IFF      DEFIND=ADDTØ.
USES1    ADDTØ(A,B)(ADD 'A' TØ 'B')
ADDTØ.   SET    DEFIND
ADDTØ    MACRØ  ARGLST,ERRT
         USES2  ADDTØ(ARGLST)(ERRT)
         ENDM   ADDTØ
ADDTØ.   MACRO  A,B,DUMMY1,DUMMY2
         PRSEQ  (ADD 'A' TØ 'B')
         ENDM   ADDTØ
```

Now consider a later point in the text where a card appears with "ADDTØ" in the operator field. It will expand as below.

```
ADDTØ    (ALPHA,BETA)          (source card)
USES2    (ALPHA,BETA)()
CALL     ADDTØ(ALPHA,BETA)
ADDTØ.   ALPHA,BETA
PRSEQ    (ADD ALPHA TØ BETA)
PMC      ØN
REM            ADD ALPHA TØ BETA
PMC      ØFF
```

The desired expansion is achieved.

---

*for some assemblers, PRSEQ is necessary. See section 5.

5. The Print Sequence Macro, PRSEQ

In section 4., we introduced the use of a macro PRSEQ, ostensibly to save space. This macro (although not as shown there) is actually necessary to some assemblers, and is desirable in order that another level of documentation be achieved.

In fact, the system shown in section 4, will not work, as shown, on the 7040-IBMAP version 3, due to a fault in the assembler's macro skeleton scan. This fault is asserted to be corrected in version 6, but we have not yet seen evidence of this. To wit, USES is supposed to build the card image

```
PRSEQ   (CØMENT)
```

into the skeleton of "NAME'.". Such a card is indeed generated while this skeleton is being built up, but the scan in question stops at the first blank, not keeping a parentheses count. Thus, while the card generated for the skeleton is

```
PRSEQ   (ADD 'A' TØ 'B')
```

, the line which is actually entered into the skeleton is

```
PRSEQ   (ADD
```

. One can readily imagine to what grevious errors the generation of such a line leads the assembler. Such a scan error is apt to be met in many assemblers, so a way around it is important.*

Further, the kind of comment generated by the macros of section 4, is not all that one might hope for. To wit, it is not sufficient that the symbolic names of variables be plugged into the comments generated by invoking a subroutine; one would prefer to plug in an explanation of what those names stand for. Put more simply, it is desirable that the expansion of comments should be a recursive process - going down through each word of the expansion (expanding it in turn, etc.) until all elements have been expanded to some bottom level.

---

*It actually suffices, for our assembler, to build the body of PRSEQ, as shown, into USES1's definition of NAME'.

This aim can be accomplished, and its accomplishment bypasses any trouble that assembler scan routines may have about blank characters. In the accomplishment, all definitions given in section 4, except that of PRSEQ and USES2, stand as given. The use of USES, on the other hand, changes. Specifically, one separates each of the words in the comment skeleton by commas.

Thus, instead of writing

```
USES    ADDTØ(A,B)(ADD 'A' TØ 'B')
```

one writes

```
USES    ADDTØ(A,B)(ADD,A,TØ,B)
```

The definition of PRSEQ, however, becomes far more complicated. It becomes necessary to add words, one at a time, to a line image. USES2 becomes

```
USES2   MACRØ   NAME(ARGLST)ERRT
        CALL    NAME(ARGLST)ERRT
        PRSET   SENT.
        NAME'.  ARGLST,ERRT
        PRLINE  (,)
        ENDM    USES2
```

Of the two additions, "PRSET SENT." sets up printing routines to use a sentence type of format, while "PRLINE (,)" causes the printing of a final line image with a comma ending the line. The generated macro "NAME'." still invokes PRSEQ, sending the comment (with actual arguments substituted) as a single argument. (In our example, "ADDTØ." would generate "PRSEQ (ADD,ALPHA,TØ,BETA)".)

PRSEQ, itself, merely does an indefinite iteration on the sub-arguments of its one argument - sending each of them on to a high level concatenate macro, PRADD.

The high level concatenate macro merely tests each of its arguments received for further expandibility. If it is so expandable, then this is done; elsewise, it passes the argument on to an exact concatenate macro, PREADD.

This concatenate macro, PREADD, is the heart of the entire automatic documentation system. It is the accessible

member of a pair of macros invented by Ann L. Locicero [1], which makes it possible to build up a linear image from successive calls of a macro.

For purposes of explanation, let us imagine that the concatenate macro were spelled "CØNCA" while its partner were spelled "CØNCAT".

The original definitions would be

```
CØNCA    MACRØ    TEXT
         CØNCAT   (TEXT)
         ENDM     CØNCA

CØNCAT   MACRØ    TEXT
CØNCA    MACRØ    MØRTXT
         CØNCAT   (TEXT,MØRTXT)
         ENDM     CØNCA
         ENDM     CØNCAT
```

Each invocation of CØNCA invokes CØNCAT so as to build its existant argument into all future calls. For instance, imagine the above definitions to be given and then the source cards

```
CØNCA    A
CØNCA    B
CØNCA    C
```

to appear. The complete expansion would be

```
         CØNCA    A
         CØNCAT   (A)
CØNCA    MACRØ    MØRTXT          Orig. meaning of CØNCA
         CØNCAT   (A,MØRTXT)        established meaning
         ENDM     CØNCA             2 of CØNCA.
         CØNCA    B
         CØNCAT   (A,B)
CØNCA    MACRØ    MØRTXT          Meaning 2 of CØNCA
         CØNCAT   (A,B,MØRTXT)      established meaning
         ENDM     CØNCA             3 of CØNCA.
         CØNCA    C
         CØNCAT   (A,B,C)
```

---

[1] Locicero, A. L.: "Linear Accumulation in Sequential Macros"; unpublished, Bellcomm, Inc.

```
CØNCA   MACRØ   MØRTXT              Meaning 3 of CØNCA
        CØNCAT  (A,B,C,MØRTXT)       established meaning
        ENDM    CØNCA                four of CONCA.
```

In actual practice, the partner macro also does bookkeeping to make sure that maximum legal argument length is not exceeded. This tends to make for an excessive amount of complication and could presumably be avoided in more adequate macro assemblers.

The PRSET routine uses its parameter to chose one of a set of actual print out macros used by PRLINE. It essentially sets a switch through the use of the operator-synonym operator.

The print line macro, PRLINE, sends its argument (if any) along to PREADD (properly parenthesized so that a separator comes out as a substitutable argument at the bottom level). It then invokes a print current line image macro, PR5., to cause the actual print. PR5. is also used by the concatenate partner macro, PR8., in case of line overflow.

The print current line image macro, PR5., changes the meaning of the concatenate partner, PR8., to that of a prepare for actual print macro, PR6., invokes the concatenate macro (which now invokes PR6. with all of its previous arguments built into its parameter), and then restores the concatenate macro and its partner to their initial meanings. The prepare for actual print macro, PR6., receives the built up line image as a single parameter (held together by its parentheses) from the prevertedly used concatenate macro, and sends it along through the dummy macro (which has been set by PRSET to be synonymous with one of the actual print macros). As PR6. has not parenthesized its single argument, this now comes apart into its several components which can be spaced apart from each other by the actual print macro (which always devolves into turning on macro print out, generating some remark card, and turning off macro printout).

Some complication could be done away with here if only it were possible to inbed blanks in the remark skeleton in the first place. Those having assemblers with such capability will find it easy to simplify the example.

6. Conditionals, Go To Statements, and Other Declarations

ALGØL-like conditional statements are easily put into an assembler in any number of ways. The if-then-else package used in the preceeding example is an off-the-cuff version and could stand considerable improvement, particularly in the recognition of single word alternatives.

As a generator of documentation, it is an easy matter to have the various relational macros directly generate a remark line, plugging in variable names. By making use of the print package, it is possible to expand the names of the variables, exactly as is done in generating subroutine comments.

The assembly operations of the package are perfectly straightforward. The IF macro merely turns on created symbols and invokes an appropriate relation macro, such as GTR. This turns off created symbols, prints a comment, creates an appropriate comparison triplet, and sets up meanings for THEN & ELSE.

The design objective was that the third member of the comparison triplet never have to contain a comparison transfer. Any assembly reached by the other alternative is to be remote. The particular system uses the location counter choosing facilities available in IBSYS, but any remote facility could serve as well. The use of the macros USEN and USEP is purely for conformity with other standards in use where this package was written. (These two macros make up for the fact that USE X and USE PREVIØUS fail to operate in a push down fashion.) If-Then-Else allows specifying either the arithmetic or logical accumulator but does not presently guard against trying to load one from the other.

The GØTØ macro is thrown in merely to allow automatic commentary. AS LABEL declarations (unlike USES & VAR) need not precede the use of the associated identifiers, there is no guarantee that GØTØ will expand the name of the destination. (It will expand it if and only if it is a transfer backwards.) "GØTØ" is used rather than redefining "TRA" because the present automatic documentation facilities cannot handle indexed addresses.

LABEL and VAR are essentially dimensioning operators. The first defines a symbol and an associated comment, and

produces that comment as a paragraph heading, followed by a colon (".."). VAR remotely assigns a full word variable and associates a commentary name with its symbolic name.

If a symbol which has been defined by either of these appears as a parameter, then its expanded name will appear in the generated commentary.

7. Conclusions

The example shown in the introduction shows that a high grade of automatic documentation is possible. However, it is not practical at this level, for one runs into limitations on table sizes.

Programs using this full level of expanded documentation run into overflows at between two and fifteen pages of expansion, depending on content.

Two possible courses of action are open: one can settle for the level of documentation generated in section 4., or one can move to repair the assembler deficiencies which cause overflow.

The section 4. level of documentation is marginally adequate. If name lists are particularly small or orderly, or if documentation requirements elsewise do not require explanation of all operands with each use, then the documentation created at this level is entirely satisfactory. The generation is extremely simple and cheap, and gives no trouble with table sizes. At almost no cost, and at some lessening of normal programmer effort, one can assure that documentation is always accurate and current. Others will find that this grade of documentation is just not good enough.

As to repairing the assembler characteristics which make the full course impractical, there are three things which would help a bit, and one which would entirely suffice:

Proper scan facilities in the building of macro skeletons would allow the expansion of comments to be done in many less steps - resulting in considerable saving in the number of skeletons generated.

Proper handling of end of line overflow on generated remark cards would relieve the macro package of considerable bookkeeping bother.

Greater table sizes would help a bit.

This last measure, however, would only be a small stop-gap. What is needed is some way to clear the tables of entries no longer needed.

While one could evolve all sorts of complicated schemes for this, the ideal solution seems to be the adoption of an ALGØL - like block structure - at least for macro definitions. In this structure, one could define BEGIN-END pairs (which could be nested). Every time a BEGIN was encountered, the condition of the tables could be marked. On meeting the corresponding END, the condition could be restored. This can be implemented merely by the addition of two words to each macro skeleton (with the addition of some elsewise vacuous skeletons for operator-synonym operations).

1031-WMK-mat

W. M. Keese, Jr.

Attachment
Appendix

APPENDIX

The following pages include the macro package used in the generation of the example shown in the introduction.

```
$TEXT  WRITE

       *                AUTOMATIC DOCUMENTATION   WM KEESE   8/15/64
       *
       *                                                          USEN-USEP
       *
       F0.     MACRO                      INITIAL RESTORATION MACRO..
        F1.    OPSYN    F0.                   RESTORE SELF TO INITIAL,
       USEP    OPSYN    F.USE                 SET 'USEP' TO GIVE FAULT,
               USE                            USE BLANK LOCATION COUNTER.
               ENDM     F0.
       *
        F1.    OPSYN    F0.
       *
       F2.     MACRO    X                USED TO HOLD REDEFINITIONS OF F1. TO 1 LV
       F1.     OPSYN    X                     RESTORE TRACE FORWARD MACRO,
       USEP    OPSYN    X                     SET 'USEP' TO CURRENT USE NAME,
               USE      X                     USE CURRENT USE NAME LOCATION CNTR.
               ENDM     F2.
       *
       F.USE   MACRO                      ILLEGAL 'USEP' FAULT..
               PMC      ON
               SPACE    1
               REM      **************************************************
               REM      *ATTEMPT TO USE -USEP- WITH NO -USEN- IN EFFECT- IGNORED*
               REM      **************************************************
               SPACE    1
               PMC      OFF
               ENDM     F.USE
       *
       USEN.   MACRO    CL               WORKING USE NEW MACRO..
       CL      OPSYN    F1.                   SET CREATED LABEL FOR PERM TRACE BACK
       USEP    OPSYN    F1.                   SET 'USEP' FOR CURRENT TRACE BACK,
       *                                    UPDATE TRACE BACK MACRO
       F1.     MACRO                          BUILD CL NAME INTO MACRO TABLE
               F2.      CL                    TO REDEF 'USEP' AND RESTORE LOC CTR.
               ENDM     F1.                   (SEE F2. FOR RESTORATION.)
       *
               USE      CL                    USE NEW LOC CTR.
               ENDM     USEN.
       *
       USEN    MACRO    ANYLAB           'USE NEW' MACRO..
               ORGCRS                         UNLESS A NAME IS GIVEN, CREATE ONE.
               USEN.    ANYLAB                DO THE WORK,
               NOCRS                          TURN OFF SYMBOL CREATOR.
               ENDM     USEN
       *
       USEP    OPSYN    F.USE                 'USEP' INITIALLY PRODUCES FAULT.
       *
```

IF-THEN-ELSE

```
*
*
AAC     OPSYN   CLA             LOAD ARITHMETIC AC..
*
DEFOP   MACRO   RM,CL,CLD       BUILD LABELS + REMOTE TYPE INTO -OP.-..
OP.     MACRO   T                   MAKE -OP.(TYPE)- MEAN
        T'.'RM  CL,CLD              INVOKE 'TYPE' OF 'REMOTETYPE'.
        ENDM    OP.
        ENDM    DEFOP
*
ELSE    MACRO                   ELSE..
        PMC     ON                  PRINT REMARK,
        REM                           * ELSE
        PMC     OFF
        OP.     E                   DO OP.(ELSE).
        ENDM    ELSE
*
EQU.    MACRO   A,B,CLT,CLD     EQUAL TO..
        NOCRS
        PRSTAT  FLAG.(IF,A(=)B)     PRINT REMARK,
        REL     LAC,LAS,*+2,CLT,A,B  CREATE LOAD AND COMPARE TRIPLET,
        DEFOP   RT,CLT,CLD          BUILD LABELS AND 'REMOTE THEN' IN OP.
        ENDM    EQU.
*
E.RE    MACRO   CLF,CLD         ELSE FOR REMOTE ELSE..
..E     SET     1                   MARK 'ELSE' DONE,
        USEN                        START NEW LOCATION COUNTER,
CLF     EQU     *                   DEFINE FALSE LABEL.
        ENDM    E.RE
*
E.RT    MACRO   CLT,CLD         ELSE FOR REMOTE THEN..
        TRA     CLD                 GENERATE TRANSFER TO END OF COND.
        USEP                        USE PREVIOUS LOCATION COUNTER.
        ENDM    E.RT
*
GEQ.    MACRO   A,B,CLT,CLD     GREATER THAN OR EQUAL TO..
        NOCRS
        PRSTAT  FLAG.(IF,A,GREATER,THAN,OR(=)B) PRINT REMARK,
        REL     ACC,CAS,CLT,CLT,A,B CREATE LOAD AND COMPARE TRIPLET,
        DEFOP   RT,CLT,CLD          BUILD LABELS AND REMOTE THEN IN OP.
        ENDM    GEQ.
*
GTR.    MACRO   A,B,CLT,CLD     GREATER THAN..
        NOCRS
        PRSTAT  FLAG.(IF,A,GREATER,THAN,B) PRINT REMARK,
        REL     AAC,CAS,CLT,*+1,A,B CREATE LOAD AND COMPARE TRIPLET,
        DEFOP   RT,CLT,CLD          BUILD LABELS AND REMOTE THEN INTO OP.
        ENDM    GTR.
```

IF-THEN-ELSE

```
.
.
IF      MACRO   A,REL,B         IF..
..E     SET     0                       MARK 'ELSE' UNDONE,
        ORGCRS                          TURN ON CREATED SYMBOLS TO CREATE
        REL'.   A,B                     2 LABELS AND DO APPROPRIATE RELATION,
        NOCRS                           MAKE SURE CREATED SYMBOLS OFF.
        ENDM    IF
.
LAC     OPSYN   CAL             LOAD LOGAICAL AC..
.
LEQ.    MACRO   A,B,CLE,CLD     LESS THAN OR EQUAL TO..
        NOCRS
        PRSTAT  FLAG.(IF,A,LESS,THAN,OR,(=)TO,B)  PRINT REMARK,
        REL     AAC,CAS,CLE,*+1,A,B CREATE LOAD AND COMPARE TRIPLET,
        DEFOP   RE,CLE,CLD              BUILD LABELS AND REMOTE ELSE INTO OP.
        ENDM    LEQ.
.
LOAD    MACRO   REG,X           LOAD REGISTER..
        IFF     REG=X                   IF OPERAND /= REGISTER
        REG     X                       THEN DO APPROPRIATE LOAD.
        ENDM    LOAD
.
LSS.    MACRO   A,B,CLE,CLD     LESS THAN..
        NOCRS
        PRSTAT  FLAG.(IF,A,LESS,THAN,B) PRINT REMARK,
        REL     AAC,CAS,CLE,CLE,A,B CREATE LOAD AND COMPARE TRIPLET,
        DEFOP   RE,CLE,CLD              BUILD LABELS AND REMOTE ELSE INTO OP.
        ENDM    LSS.
.
NEQ.    MACRO   A,B,CLE,CLD     NOT EQUAL TO..
        NOCRS
        PRSTAT  FLAG.(IF,A,NOT,EQUAL,TO,B) PRINT REMARK,
        REL     LAC,LAS,*+2,CLE,A,B CREATE LOAD AND COMPARE TRIPLET,
        DEFOP   RE,CLE,CLD              BUILD LABELS AND REMOTE ELSE INTO OP.
        ENDM    NEQ.
```

```
*                                                          IF-THEN-ELSE
*
*                                        CREATE LOAD AND COMPARE TRIPLET..
REL     MACRO   REG,COMP,LGTR,LEQ,A,B
        LOAD    REG,A                    DO APPROPRIATE LOAD.
        COMP    B                        SET UP COMPARISON,
        TRA     LGTR                     IF GREATER, GO TO GREATER LABEL.
        TRA     LEQ                      IF EQUAL, GO TO EQUAL LABEL.
        ENDM    REL                      ASSUME LESS THAN CASE FOLLOWS.
*
THEN    MACRO                          THEN..
        PMC     ON                       PRINT REMARK.
        REM                                • THEN
        PMC     OFF
        OP.     T                        DO OP.(THEN).
        ENDM    THEN
*
T.RE    OPSYN   NULL                   THEN FOR REMOTE ELSE.. (= NULL)
*
T.RT    MACRO   CLT,CLD                THEN FOR REMOTE THEN..
        USEN                             USE REMOTE LOCATION COUNTER,
CLT     EQU     *                        DEFINE TRUE LABEL.
        ENDM    T.RT
*
..RE    MACRO   CLE,CLD                END OF CONDITIONAL FOR REMOTE ELSE..
        IFT     ..E=0                    IF THERE WAS NO 'ELSE'
CLE     EQU     CLD                      DEFINE ELSE-LOC = DONE-LOC.
        IFF     ..E=0                    IF THERE WAS AN 'ELSE'
        TRA     CLD                      TRNSFER OUT OF THE REMOTE,
        IFF     ..E=0
        USEP                             AND END THE REMOTE SECTION.
CLD     EQU     *                        DEFINE END OF COND. LABEL.
        ENDM    ..RE
*
..RT    MACRO   CLT,CLD                END OF CONDITIONAL FOR REMOTE THEN..
CLD     EQU     *                        DEFINE END OF COND. LABEL.
        ENDM    ..RT
*
...     MACRO                          END OF CONDITIONAL..
        PMC     ON                       PRINT REMARK,
        REM                                • .....
        PMC     OFF
        OP.     .                        DO OP.(END OF CONDITIONAL).
        ENDM    ...
```

```
*PRINT MACROS                                                    PRINT
*
*                  THESE MACROS ARE USED TO BUILD UP AND PRINT LINE
*               IMAGES IN THE AUTOMATIC GENERATION OF COMMENTS.
*
*                    THE MAIN 'ENTRIES' ARE
*
*       PRADD                           APPEND PHRASE TO LINE
*       PREADD                          APPEND EXACT PHRASE TO LINE
*       PRLINE  (ANY PUNCTUATION)       PRINT LINE
*       PRSEQ   (PHRASE SEQUENCE)       APPEND SEQUENCE OF PHRASES
*       PRSTAT  TYPE(MESAGE)PUNCT       PRINT MESAGE AS TYPE WITH PUNCT
*       PRSET   TYPE                    SET LINE TYPE
*
*               WHERE 'TYPE' STANDS FOR EITHER 'HEAD.', 'FLAG.',
*               OR 'SENT.'.
*
*                  THE OTHER INTERNALLY USED PARTS ARE
*
*       PR1.                            ACTUAL HEADER PRINT
*       PR2.                            ACTUAL FLAGGED PRINT
*       PR3.                            ACTUAL SENTENCE PRINT
*       PR4.                            ACTUAL PRINT DUMMY
*       PR5.                            PRINT CURRENT LINE
*       PR6.                            STRIP PARENS FOR ACTUAL PRINT
*       PR7.                            NORMAL APPEND PARTNER
*       PR8.                            APPEND PARTNER
*       PR11.                           SET HEADER PRINTING
*       PR12.                           SET FLAGGED PRINTING
*       PR13.                           SET SENTENCE PRINTING
*
*
PR1.    MACRO   A,B,C,D,E,F,G,H ACTUAL HEADER PRINT..
        PMC     ON
        REM                                 'A' 'B' 'C' 'D' 'E' 'F''G'
        PMC     OFF
        ENDM    PR1.
*
PR2.    MACRO   A,B,C,D,E,F,G,H ACTUAL FLAGGED PRINT..
        PMC     ON
        REM                                   'A' 'B' 'C' 'D' 'E' 'F''G'
        PMC     OFF
        PR13.                               RESET PRINT OUT FOR NORMAL LINES.
        ENDM    PR2.
*
PR3.    MACRO   A,B,C,D,E,F,G   ACTUAL SENTENCE PRINT..
        PMC     ON
        REM                                     'A' 'B' 'C' 'D' 'E''F'
        PMC     OFF
        ENDM    PR3.
*
PR4.    OPSYN   PR2.            ACTUAL PRINT DUMMY..
*                                   (SWITCHED BY PRSET)
```

```
*                                                                PRINT
*
PR5.    MACRO                   PRINT CURRENT LINE..
 PR8.   OPSYN   PR6.                CHANGE APPEND PARTNER TO MEAN
*                                   STRIP PARENS FOR ACTUAL PRINT
        PREADD                     AND INVOKE IT.
PRWCT.  SET     0                  RESET WORD COUNT.
 PR8.   OPSYN   PR7.                RESET APPEND PARTNER.
PREADD  OPSYN   PR8.                RESET APPEND EXACT WORD.
        ENDM    PR5.
*
PR6.    MACRO   TEXT            STRIP PARENS FOR ACTUAL PRINT..
        PR4.    TEXT                SEND SEPARATED ARGS THROUGH
        ENDM    PR6.                ACTUAL PRINT DUMMY.
*
PR7.    MACRO   TEXT            NORMAL APPEND PARTNER..
PREADD  MACRO   MORTXT              REDEFINE APPEND EXACT WORD
        PR8.    (TEXT,MORTXT)       TO INVOKE PARTNER WITH TEXT BUILT IN.
        ENDM    PREADD
PRWCT.  SET     PRWCT.+1            UP WORD COUNT.
        IFT     PRWCT.=PRWLM.       IF WORD COUNT = WORD LIMIT
        PR5.                        THEN PRINT CURRENT LINE.
        ENDM    PR7.
*
PR8.    OPSYN   PR7.            APPEND PARTNER..(SET TO NORMAL)
*
PR11.   MACRO                   SET FOR HEADER LINES..
 PR4.   OPSYN   PR1.                MAKE-ACT. PRINT--ACT. HEAD. PRINT-,
PRWLM.  SET     6                  6 =. WORD LIMIT.
        ENDM    PR11.
*
PR12.   MACRO                   SET FOR FLAGGED PRINT..
 PR4.   OPSYN   PR2.                MAKE ACTUAL PRINT = FLAGGED PRINT.
PRWLM.  SET     6                  6 =. WORD LIMIT.
        ENDM    PR12.
*
PR13.   MACRO                   SET FOR SENTENCE LINES
 PR4.   OPSYN   PR3.                MAKE-ACT. PRINT--ACT.SENTENCE PRINT-,
PRWLM.  SET     5                  5 =. WORD LIMIT.
        ENDM    PR13.
*
PR14.   MACRO   OP
        OP
        ENDM    PR14.
```

```
.
.
PRADD   MACRO   TEXT            ADD PHRASE TO LINE..
        IFT     DEFIND=TEXT'.       IF TEXT IS DEFINED
        PR14.   TEXT'.              THEN EXPAND IT
        IFF     DEFIND=TEXT'.       ELSE
        PREADD  (TEXT)              ADD IT DIRECTLY TO LINE.
        ENDM    PRADD
.
PREADD  OPSYN   PR8.            APPEND EXACT PHRASE TO LINE..
.
PRLINE  MACRO   PUNCT           PRINT LINE..
PRWLM.  SET     PRWLM.+1            UP WORD LIMIT FOR PUNCTUATION,
        PREADD  ((PUNCT))           ADD ANY PUNCTUATION,
PRWLM.  SET     PRWLM.-1            RESTORE WORD LIMIT,
        PR5.                        PRINT CURRENT LINE IMAGE.
        ENDM    PRLINE
.
PRSET   MACRO   TYPE            SET LINE TYPE..
        IFT     TYPE=HEAD.          (DECODES -TYPE- AND
        PR11.                       CALLS APPROPRIATE INITIALIZER.)
        IFT     TYPE=FLAG.
        PR12.
        IFT     TYPE=SENT.
        PR13.
        ENDM    PRSET
.
PRSEQ   MACRO   COMENT          APPENT PHRASE SEQUENCE..
        IRP     COMENT
        PRADD   (COMENT)            APPEND PHRASE TO LINE.
        IRP
        ENDM    PRSEQ
.
PRSTAT  MACRO   TYPE,MESAGE,PUNCT
        PRSET   TYPE
        PRSEQ   (MESAGE)
        PRLINE  (PUNCT)
        ENDM    PRSTAT
```

```
*                                                              USES
GOTO    MACRO   LABEL
        TRA     LABEL
        PRSET   SENT.
        PREADD  GO
        PREADD  TO
        PRADD   LABEL
        PRLINE  (,)
        ENDM    GOTO
*
LABEL   MACRO   NAME,COMENT
        PMC     ON
NAME    EQU     *
        PMC     OFF
NAME'.  SET     DEFIND          MARK NAME AS BEING DEFINED.
NAME'.  MACRO                   DEFINE ASSOCIATED EXPANSION MACRO..
        PRSEQ   (COMENT)        PRINT OUT PHRASES IN MACRO.
        ENDM    NAME'.
        PRSET   HEAD.           SET PRINTER TO PRINT HEADING.
        NAME'.                  PUT COMMENT PHRASES IN LINE IMAGE,
        PRLINE  (..)            PRINT LINE FOLLOWED BY COLON.
        ENDM    LABEL
*
USES    MACRO   SR(ALIST)COMENT
        IFF     SR'.=DEFIND     IF SUBROUTINE MACRO NOT DEFINED
        USES1.  SR(ALIST)(COMENT)  THEN DEFINE IT
        ENDM    USES
*
USES1.  MACRO   SR(ALIST)COMENT
 SR'.   SET     DEFIND          NOTE THAT SR IS DEFINED.
 SR     MACRO   ARGLST,ERRT     DEFINE SR NAME
        USES2.  SR(ARGLST)(ERRT)  TO INVOKE STANDARD CALL.
        ENDM    SR
 SR'.   MACRO   ALIST,DUMY1,DUMY2  DEFINE AUGUMENTED NAME
        PRSEQ   (COMENT)        TO APPEND ALL COMMENT WORDS.
        ENDM    SR'.
        ENDM    USES1.
*
USES2.  MACRO   SR(ARGLST)ERRT
        CALL    SR(ARGLST)ERRT  PRODUCE DESIRED STD. CALL,
        PRSET   SENT.           READY PRINT MACROS FOR SENTENCE.
        SR'.    ARGLST,ERRT     PUT COMMENT WORDS IN LINE IMAGE,
        PRLINE  (,)             PRINT LINE WITH COMMA.
        ENDM    USES2.
*
VAR     MACRO   NAME,COMENT
        USEN    STOR
NAME
        USEP
NAME'.  SET     DEFIND
NAME'.  MACRO
        PRSEQ   (COMENT)
        ENDM    NAME'.
        ENDM    VAR
```

```
                *
                *
                DEFSYM MACRO   LIST
                       IRP     LIST
                 LIST  SET     DEF.
                 DEF.  SET     DEF.+1
                       IRP
                       ENDM    DEFSYM
37200            DEF.  SET     16000
                *
                       DEFSYM  (AAC.,DEFIND.,LAC)
                       DEFSYM  (HEAD.,FLAG.,SENT.)
                *
                *
00001           I      SET     1
                *
00000 10000 APPND
00000 10000 GNC
00000 10000 GNBC
00000 10000 HEAD
00000 10000 NONSPC
00000 10000 SAVID
                       NOCRS
```

```
*       ONE HEADS EACH ROUTINE OR SEGMENT WITH A 'USES' LIST,E.G.
*
        USES    APPND(CHAR)(APPEND,CHAR,TO,ID,STRING)
        USES    GNC()(GET,NC)
        USES    GNBC()(GET,NON,BLANK,CHARACTER)
        USES    SAVID()(SAVE,IDENTIFIER,STATUS)
*
*       WITH VARIABLE DECLARATIONS OF THE FORM
*
        VAR     NC(NEXT,CHARACTER)
        VAR     PC(PEEK,CHARACTER)
*
*               THE TEXT OF THE PROGRAM MAY THEN BE WRITTEN
*       IN ABBREVIATED FORM, AS BELOW, IN THE SOURCE DECK.
*
*       LABEL   ANAME(APOSTROPHE-ENCLOSED,NAME)
*       GNC
*       IF      NC,NEQ(='H00000')
*       THEN
*       GOTO    ANAME
*       ELSE
*       APPND   NC
*       GNC
*       GOTO    NONSPC
*       ...
*
*       LABEL   BLNKF(BLANK,FOUND)
*       APPND   PC
*       LAC     HEAD,I                  SET POINTER TO HEAD OF TREE.
*       GNBC
*       TRA     0,I                     TRACE SPECIAL WORD TREE.
```

```
                *                THIS WILL BE EXPANDED IN THE LISTING AS IS SHOWN HERE.
                *
                *
                        LABEL   ANAME(APOSTROPHE-ENCLOSED,NAME)
00006           ANAME   EQU *
                                                APOSTROPHE-ENCLOSED NAME ..
                        GNC
                                                  GET NEXT CHARACTER ,
                        IF      NC,NEQ(=H00000')
                                                * IF NEXT CHARACTER NOT EQUAL TO
                                                  =H00000'
                        THEN
                                                * THEN
                        GOTO    ANAME
                                                  GO TO APOSTROPHE-ENCLOSED NAME ,
                        ELSE
                                                * ELSE
                        APPND   NC
                                                  APPEND NEXT CHARACTER TO ID
                                                  STRING ,
                        GNC
                                                  GET NEXT CHARACTER ,
                        GOTO    NONSPC
                                                  GO TO NONSPC ,
                        ...
                                                * .....

                *
                        LABEL   BLNKF(BLANK,FOUND)
00016           BLNKF   EQU *
                                                BLANK FOUND ..
                        APPND   PC
                                                  APPEND PEEK CHARACTER TO ID
                                                  STRING ,
00003 10001             LAC     HEAD,I          SET POINTER TO HEAD OF TREE,
                        GNBC
                                                  GET NON BLANK CHARACTER ,
00000 10000             TRA     0,I             TRACE SPECIAL WORD TREE.
```